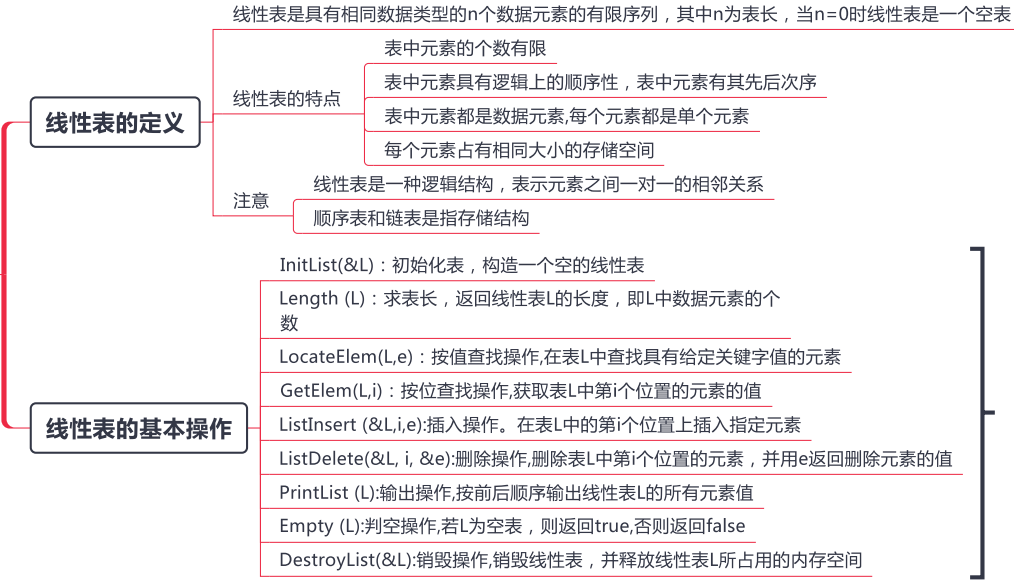


2.1 线性表的定义和基本操作



考试尽量使用这些函数名称，方便老师阅卷

2.2 线性表的顺序表示

顺序表的定义

线性表的顺序存储又称顺序表

用一组地址连续的存储单元依次存储线性表中的数据元素

逻辑上相邻的两个元素在物理位置上也相邻

静态分配 数组的大小和空间已经固定，一旦空间占满，再加入新的数据将会产生溢出，程序就会崩溃

动态分配 存储数组的空间是在程序执行过程中通过动态存储分配语句分配的

一旦数据空间占满，就另外开辟一块更大的存储空间，用以替换原来的存储空间

注意 动态分配仍然是顺序存储结构，物理结构没有变化，依然是随机存取方式，只是分配的空间大小可以在运行时决定

随机访问，即通过首地址和元素序号可在时间O(1)内找到指定的元素

特点 存储密度高，每个结点只存储数据元素 与链表形成对比

逻辑上相邻的元素物理上也相邻，当执行插入和删除操作时，需要移动大量元素

顺序表上基本操作的实现

插入操作

由于顺序表的中元素的物理位置是相邻的，所以当插入新元素的时候就需要对表中的元素进行整体移动

最好情况：在表尾插入(i = n+1),元素后移语句将不执行，时间复杂度为O(1)

最坏情况：在表头插入(i=1),元素后移语句将执行n次，时间复杂度为O(n)

假设 p_i ($p_i = 1/(n+1)$)是在第i个位置上插入一个结点的概率

则在长度为n的线性表中插入一个结点时，移动节点的平均次数为

$$\sum_{i=1}^{n+1} p_i(n-i+1) = \sum_{i=1}^{n+1} \frac{1}{n+1}(n-i+1) = \frac{1}{n+1} \sum_{i=1}^{n+1} (n-i+1) = \frac{1}{n+1} \frac{n(n+1)}{2} = \frac{n}{2}$$

时间复杂度为 O(n)

删除操作

即为移动元素，对想要删除的元素进行覆盖

最好情况：删除表尾元素(i=n)，无须移动元素，时间复杂度为O(1)

最坏情况：删除表头元素(i=1),需移动除第一个元素外的所有元素，时间复杂度为O(n)

假设 p_i ($p_i = 1/n$)是在第i个位置上删除一个结点的概率

则在长度为n的线性表中删除一个结点时，删除节点的平均次数为

$$\sum_{i=1}^n p_i(n-i) = \sum_{i=1}^n \frac{1}{n}(n-i) = \frac{1}{n} \sum_{i=1}^n (n-i) = \frac{1}{n} \frac{n(n-1)}{2} = \frac{n-1}{2}$$

时间复杂度为 O(n)

按值查找(顺序查找)

最好情况:查找的元素就在表头,仅需比较一次,时间复杂度为 O(1)

最坏情况:查找的元素在表尾(或不存在)时,需要比较n次,时间复杂度为O(n)

假设 p_i ($p_i=1/n$)是查找的元素在第i ($1 \leq i \leq L.length$)个位置上的概率

则在长度为n的线性表中查找值为e的元素所需比较的平均次数为

$$\sum_{i=1}^n p_i \times i = \sum_{i=1}^n \frac{1}{n} \times i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

时间复杂度为O(n)

2.3线性表的链式表示(上)

背景

顺序表

优点：可以随时存取表中的任意一个元素
缺点：插入和删除操作需要移动大量元素

链式存储线性表

优点：不需要使用地址连续的存储单元，插入和删除操作不需要移动元素，而只需修改指针
缺点：失去顺序表可随机存取的优点

实现方式

带头结点

空表判断：L=NULL。代码书写不便

不带头结点

空表判断：L->next=NULL。写代码更方便

单链表的定义

线性表的链式存储又称单链表，它是指通过一组任意的存储单元来存储线性表中的数据元素

对每个链表结点，除存放元素自身的信息外，还需要存放一个指向其后继的指针

结构

data：数据域,存放数据元素

next：指针域,存放其后继结点的地址

优点

解决顺序表需要大量连续存储单元的缺点

单链表附加指针域，浪费存储空间

缺点

查找某个特定的结点时，需要从表头开始遍历，依次查找

单链表上基本操作的实现

采用头插法建立单链表

从一个空表开始,生成新结点,并将读取到的数据存放到新结点的数据域中,然后将新结点插入到当前链表的表头,即头结点之后

示意图

时间复杂度

每个结点插入的时间复杂度为O(1)
设单链表长为n,则总时间复杂度为O(n)

优点：算法实现简单

缺点：生成的链表中结点的次序和输入数据的顺序不一致 可以利用这个特点进行链表转置

该方法将新结点插入到当前链表的表尾,为此必须增加一个尾指针 r,使其始终指向当前链表的尾结点

采用尾插法建立单链表

示意图

按序号查找结点值 时间复杂度O(n)

按值查找表结点 时间复杂度O(n)

链表的本身特点原因，查找只能依次遍历查找

插入结点操作

示意图

实现代码段

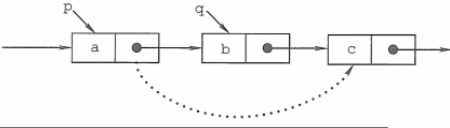
p=GetElem(L,i-1);
s->next=p->next;
p->next=s;

若在给定的节点下进行插入，则时间复杂度为O(1)

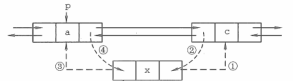
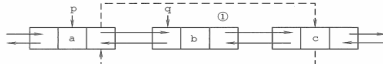
本算法主要的时间开销在于查找第i-1个元素,时间复杂度为O(n)

2.3线性表的链式表示(中)

单链表上基本操作的实现

- 对某一结点进行前插操作 即寻找到要插入结点的前一个结点, 然后按照正常插入方法执行即可
时间复杂度为 $O(n)$
- 删除结点操作 删除结点操作是将单链表的第1个结点删除。先检查删除位置的合法性, 后查找表中第1-1个结点, 即被删结点的前驱结点, 再将其删除
- 示意图 
- 该算法
- 删除结点*p 方法一 要删除某个给定结点*p, 通常的做法是从链表的头结点开始顺序找到其前驱结点, 然后再执行删除操作
算法的时间复杂度为 $O(n)$
- 方法二 删除结点*p的操作可用删除*p的后继结点操作来实现, 实质就是将其后继结点的值赋予其自身, 然后删除后继结点
时间复杂度为 $O(1)$ 。
- 求表长操作 从第一个结点开始顺序依次访问表中的每个结点, 为此需要设置一个计数器变量, 每访问一个结点, 计数器加1, 直到访问到空结点为止
算法的时间复杂度为 $O(n)$

双链表

- 单链表的缺点 单链表只能从头结点依次顺序地向后遍历
不能够快速的对其前驱结点进行操作
- 概念 双链表结点中有两个指针prior和next, 分别指向其前驱结点和后继结点
- 双链表的插入操作 示意图 
- 原则就是在进行双链表插入的时候要保证不会造成断链
- 双链表的删除操作 示意图 
- 原则仍然是在进行操作的时候一定要考虑是否会造成断链

操作的时候一定要考虑, 本操作顺序是否会造成后继结点的丢失

循环链表

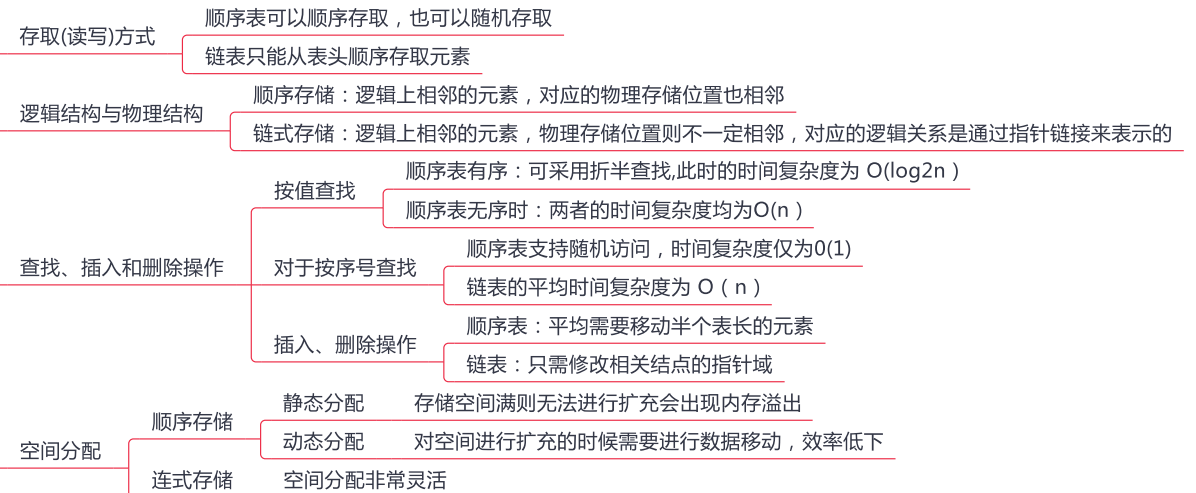
- 循环单链表 循环单链表和单链表的区别在于, 表中最后一个结点的指针不是NULL, 而改为指向头结点, 形成了一个环
- 判空条件: 头结点的指针是否等于头指针
- 仅设尾指针 如果是要在链首之前插入结点, 此时效率明显更高
时间复杂度 $O(1)$
- 循环双链表 在循环双链表中, 头结点的 prior 指针还要指向表尾结点
- 在循环双链表L中, 某结点*p为尾结点时, $p \rightarrow next = L$
- 当循环双链表为空表时, 其头结点的 prior 域和 next 域都等于L

静态链表

- 基本结构 静态链表借助数组来描述线性表的链式存储结构 静态链表也要预先分配一块连续的内存空间
- 结点也有数据域data和指针域next 这里的指针是结点的相对地址(数组下标), 又称游标
- 特点 静态链表以 $next = -1$ 作为其结束的标志
- 静态链表的插入、删除操作与动态链表的相同, 只需要修改指针, 而不需要移动元素
- 优点: 增、删 操作不需要大量移动元素
- 缺点: 不能随机存取, 只能从头结点开始依次往后查找; 容量固定不可变
- 适用场景: ①不支持指针的低级语言; ②数据元素数量固定不变的场景 (如操作系统的文件分配表FAT)

2.3线性表的链式表示(下)

顺序表和链表的比较



选取存储结构

